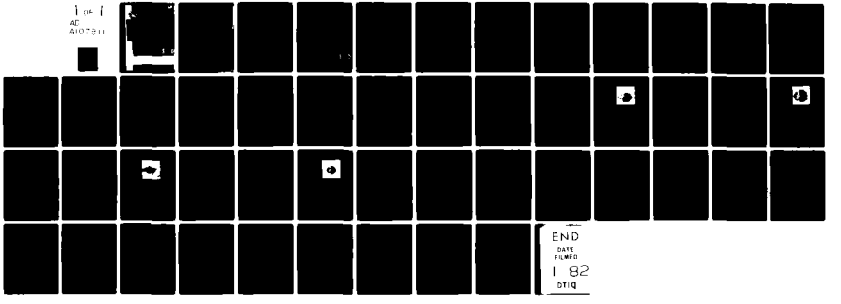
AD-A107 911 OHIO STATE UNIV RESEARCH FOUNDATION COLUMBUS F/G 9/2
TERRAIN MODEL ANIMATION.(U)
SEP 81 J L BOOKER, C CSURI, R MARSHALL N61339-80-C-0008
UNCLASSIFIED NAVTRAEQUIPC-80-C-0008-1 NL



END
DATE FILMED
1 82
DTIC

Technical Report: NAVTRAEQUIPCEN 80-C-0008-1

LEVEL II

TERRAIN MODEL ANIMATION

CHARLES CSURI
ROBERT MARSHALL
ROGER WILSON
The Ohio State University Research Foundation
Columbus, Ohio 43212

and

JOHN L. BOOKER
Naval Training Equipment Center
Orlando, FL 32813

SEPTEMBER 1981

FOR PERIOD DECEMBER 1979 - OCTOBER 1980

NAVAL TRAINING EQUIPMENT CENTER

ORLANDO, FLORIDA 32813

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(12) 48

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: NAVTRAEQUIPCEN 80-C-0008-1	2. GOVT ACCESSION NO.: AD-A1079-11
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): TERRAIN MODEL ANIMATION	5. TYPE OF REPORT & PERIOD COVERED: Final 12/1/79-10/15/80
	6. PERFORMING ORG. REPORT NUMBER: NAVTRAEQUIPCEN 80-C-0008-1
7. AUTHOR(s): John Booker, Charles Csuri, Robert Marshall and Roger Wilson	8. CONTRACT OR GRANT NUMBER(s): N61339-80-C-0008
9. PERFORMING ORGANIZATION NAME AND ADDRESS: The Ohio State University Research Foundation, 1314 Kinnear Road, Columbus, Ohio 43212	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: DEPARTMENT OF THE NAVY, Naval Training Equipment Center, Orlando, Florida 32813	12. REPORT DATE: September 1981
	13. NUMBER OF PAGES: 46
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

procedure models, terrain model, data generation, flight simulation, computer graphics, computer animation, Z-buffer, display algorithm

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Procedure models are used to combine fundamental data elements in the creation of unified objects comprising the terrain model. A procedure model to generate trees of various species was implemented. Interactive techniques were developed to generate mountains. An analysis was made of the performance of a Z-buffer display algorithm. Test results are included in the report.

DD FORM 1 JAN 73 1473

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

TASK OVERVIEW AND PERFORMANCE

INTRODUCTION

The Computer Graphics Research Group at The Ohio State University has undertaken a task which involves generation and display of a data base. This data base depicts a terrain model, the imagery of which is derived from a small model board supplied by the Navy. To be more specific, the terrain includes trees (26), flat land, a small river, one house, one barn, one car, one bridge, a road, telephone poles and a mailbox. The data base will be utilized for production of animation simulating a pilot's point of view flying at low altitudes. The progress has been documented with Polaroid photograph's as well as 35 mm color slides which illustrate multiple views of the terrain model along with changes in scale and color.

Among the papers published by the Computer Graphics Research Group in the past year or so are two papers entitled "Toward an Interactive High Visual Complexity Animation System" and "Procedure Models for Generating Three Dimensional Terrain." The research undertaken by the group during the completion of these two papers has enabled us to develop techniques which were applied to this Navy project.

The unique problems presented by this project are at once diversified and yet interdependent. In the development of a digital data base of specified terrain, alogrithms and an animation system must be adapted or developed. Filming and taping routines must be perfected as well as procedures for snooting slides and photographs. The methods of data generation for this project are gleaned from different sources in the group as are the methods of designating color, complexity of the imagery, and description of motion. This paper describes, at length, the methods by which the group approached the various problems of the project and how the solutions were implemented.

ANTS

Before work could begin on the project, a number of software tools had to be developed. The VAX was functioning with operating system intact, but an animation language (ANTS) had yet to be implemented. This language was developed by Ronald Hackathorn, Richard Parent, Robert Marshall, Wayne Carlson and Marc Howard.

This system facilitates the generation, manipulation, and display of highly detailed data with the aid of interactive devices and a video interface connected to a high resolution RGB monitor. The system enables the animator to create a variety of objects (including texture) and to specify the necessary transformation for an animation sequence. A run-length processing technique, combined with a brute force Z-buffer algorithm, has been designed and implemented that can handle the intersection of several million faces, lines, and points.

This capability is necessary for displaying the complex imagery involved in creating a terrain model consisting of trees, buildings, etc. The system allows for the generation and display of not just thousands of edges, but rather millions.

DATA GENERATION

The CGRG data generation routine is a system designed by Dr. Richard Parent. Using a Vector General, the artist has several methods with which to construct objects needed in the animation sequence. A simple draw routine may be performed with the aid of a cursor. A 2-D drawing is made on the VG; then this drawing can be made into 3-D form by either extending the object along the Z axis or by rotating the object around the Y axis. The depth in Z as well as the number of slices around Y can be controlled by the artist by using the dials in this routine.

The option of using a geometric shape such as a cube, sphere, cylinder, or pyramid is open to the animator as well. By pushing a certain combination of function buttons, the desired object will appear on the VG screen.

In order to "build" a simple object using two or more shapes, the artist may, in the case of a simple two-shape object, combine the objects into one in the data generation routine itself. In the case of a more complex object, the animation language facilitates the combining or grouping of two or more objects into one. This may be done by simply positioning the objects next to each other or by employing "virtual intersection" (the apparent joining of two objects into one by the visible surface calculation).

The new animation system is capable of handling complex objects and therefore demands new methods of creating data. This ability to handle complex images is a primary factor when considering terrain as a source for images. Creating trees or forest lines is a capability which sets the ANTS system apart from others currently in use. When moving through a terrain in an animation sequence, the trees must be represented in a three-dimensional format so that the observer can pass by or over them. Likewise, a three-D model enables the animator to rotate the tree so that viewing from all sides is possible.

Of central interest in this project is the method for generating and displaying trees. The procedure model for a tree uses many parameters, some of which define the type of tree and others of which control its generation. Parameters which define the tree type are the leaf type, the branch structure, the color, and the limits on the size and shape of the overall tree. The other group of parameters which determine the

specific tree generated include the number of leaves in a tree, the actual size of the branches, the density of the leaves, and the height of the tree. The specific parameters are not independent. For example, the density of the leaves depends on the average size of the branches and the number of leaves. One need only specify two of the three parameters and the other will be implicit. If density and branch size parameters are specified, the number of leaves is already determined. Also, an interval for the specific parameters can be specified, with the value on the interval determined by a pseudo random number generator. This reduces the burden on the user of specifying each parameter value explicitly.

Randomness is also used in combination with the specified parameters to produce unique trees of a given species. The approximate location of each branch and leaf is determined by the model using the input parameters. The final position and orientation of each individual leaf is selected by a random perturbation of the calculated values. The underlying structure, determined by the procedure model, insures that a realistic image is produced, while the randomness gives the unique appearance of the individual trees.

One of our implemented procedure models for trees (of a general type) uses the following parameters:

a. number of leaves
b. length of each branch
c. leaf element description
d. color
e. position of tree
f. size of leaf element
g. distance between branches
h. distance between leaves
i. random number seed

Once these parameters have been specified, the degrees of variability of individual trees of this type is determined. On input of the parameters, the procedure model for a tree works as follows: the leaf elements are organized on the branch according to the number of leaves, the length of the branch, and the size of the leaf. These digital element positions are modified slightly with use of a random number generator to provide a non-uniform orientation (with respect to the rotation of the element) and final position on the branch. After the number of branches specified has been created as above, they are positioned around the trunk element. Their positions and rotations are modified in order to complete the three-dimensional tree model. Finally the tree is

positioned in the terrain model according to its input parameter, and colors are selected from the color palette to be assigned to each primitive element according to its type and orientation.

Procedural Building Routines

Programs can be written in the ANTS language which leads the user through a sequence of steps in which he/she specifies parameters sufficient for the program to construct a surface definition for one instance of a broad class of objects. While currently the ANTS language has no surface generation facilities, ANTS does have a powerful capability for combining predefined objects in order to form new complex shapes. The surface descriptions for these new objects can be directed into a file which can then be referenced in subsequent animation scripts. Alternatively, the surface description can be directed immediately to the scanning routine for inclusion in an animation sequence which is currently being calculated.

At the CGRG, programs have been written for procedurally constructing trees and buildings with windows overlaid on front, sides and back and an optional roof. The user specifies the width, depth and height for the building and a cube is appropriately scaled along the three axes. The user then selects one of several window types to be distributed on the sides of the building. The width and height of the window is specified by the user as are marginal spacings for 1) the sides of the front and back walls, 2) the sides of the side walls, 3) the top of the walls and 4) the bottom of the walls. The distribution of the window is performed by the program according to either how much space the user wants between windows or the number of windows the user wants down and across the walls.

Similarly, the user selects one of several roof types, if desired, and gives dimensions for the roof. The program then scales the roof type accordingly and places the roof on top of the building already constructed.

Once several building elements have been so constructed, facilities in ANTS allows the user to easily combine the elements to form multitiered buildings, other complex building structures (e.g., a church with steeple) or a city block consisting of multiple buildings.

GRID ROUTINE

The objects which make up the ground in the terrain model were generated on the Vector General display wing and grid routine. This routine allows the user to define rectangular areas and then warp

selected points on the grid. The buttons are used to select the functions in the interactive mode. Some of the functions move the cursor along rows or columns of the grid, enable or disable scene rotation, initialize a group wrap, finalize or clear a warp, and return to command mode (terminal input). The details are used for the rotation, and degree of wrap. In the command mode, grids may be input and output as files, new grids defined, and grids triangulated. Two important functions in this mode are cutting the grid, and adding resolution to the grid. The grid may be cut into four sections at any point inside the grid. These functions allow the user to define a grid and then cut out a position of it and double the resolution to enhance a portion of the grid. This enhanced section can then be combined with the other sections to produce a single object.

The ground on the terrain model was defined first as a flat rectangle. The group warp was used to make the depression for the river. The points along the bank were warped to lesser degrees than the bottom of the river. Later the section containing the mound was cut out and warped separately and then added to the original grid. This grid extended just past the detail of the scenes (the trees, house and barn). A b-spline routine was used on different sections to smooth the grid. The flat portion near the house and barn did not need to be smoothed. The mound was smoothed slightly and the area of the banks was smoothed the most. Several flat grids were extended to fill out to the horizon.

ANIMATION

In order to perform the required animation for this project, a method of moving the observer through the data base was sought. The normal functions in the animation language such as change position and change rotation could possibly solve the problem, but only with a maximum amount of effort and at the cost of accuracy. The term "path move" applies to a routine in which key points (X,Y,Z) are designated from the data base, then a b-spline function is performed on these points to effect a smooth path through the terrain. This is especially useful for plotting a low level flight in which the altitude of the observer is continually changing in order to move over obstacles in the terrain such as buildings and trees. Since the height of these objects is a known quantity it is a simple task for the user to plot the path through terrain and specify points on which to move the observer.

The user can achieve a variety of effects such as zooming, panning, trucking shots, etc., by moving the center of interest (COI) as well as the observer. For example, if the user wishes to simulate a zoom into the scene the COI will remain stationary as the observer moves closer to the point of interest in the scene. For a pan the observer can remain the same, as the COI is moved from side to side. Performing a trucking shot involves positioning the COI and the observer and then moving them along parallel paths. Simulating a view from a moving tank

was achieved by placing the COI some units ahead of the observer and moving them along the same path. A low level flight might be a combination of these functions.

In addition to these capabilities, the perspective or view angle may be changed by the user to determine exactly how acute the perspective should be in each scene. The default view angle on our system is 80 degrees. This was changed to 50 degrees in order to create a picture which covered a rather large area in X and Z. When default was used, the angle was so severe that the horizon line was bent and trees positioned near the edge of the scene seemed to be leaning.

COLOR SPECIFICATION

The frame buffer used in the project has a color table (palette) of 1024 locations. Each table location defines a 24 bit RGB value (8 bits each of red, green, and blue). When the objects are defined, they are assigned a color number. ANTS uses this color number to calculate the palette address of each triangle. The colors are defined in ANTS as a section of the palette. The color is assigned a starting location in the palette and a length. ANTS calculates the intensity of each triangle depending on the position of the triangle, the light source and the degree of ambiant light.

A total of 36 colors are used in the terrain model. The length of the colors in the palette varies from 1 to 64. The tree colors contain 32 levels of RGB, while other colors contain only one level (for example, the car tires in the terrain model). We currently have two ways to specify the colors. The first one is with an ANTS script. The user first defines the location and length of the new color (or the number of the color to modify). The script then prompts for the color number and the RGB value of the first location in the color and the last location in the color. The remaining RGB values are found by interpolation. This script can also modify a portion of an existing color. The other method uses a bitpad. The colors must be defined in ANTS and then the palette saved in a file. A utility program outside of ANTS uses the saved palette file in order to define colors interactively with the bitpad. There is a simple menu on the top of the bitpad area for seeing the current RGB value, assigning a new color to use, and for exit. The rest of the bitpad is used for 3 slides, one each for red, green, and blue. When the new color area is hit, the user is prompted for the color number, the mode, and the hues to change. There are 3 modes. Single mode will change only one hue in the color. Double mode will interpolate to a lower hue of the color. Triple mode interpolates between a lower and higher hue in the color. The screen shows the red, green, and blue components of the hue being changed along with the color. The bottom of the screen shows each hue of the color.

The script method is good for simple colors which vary evenly from dark to light, but it is hard to tell what the color will look like

from only the RGB value. The bit-pad method allows the image to be on the screen while the colors are changed, providing instant feedback.

The colors we first used varied from black to the lightest hue possible. This caused problems with the image during animation due to some of the small triangles. The triangles were popping in and out of view due to being too small to always scan 1 pixel, or being covered by other triangles. When these triangles were assigned the lighter hues of the color they flashed. This is very noticeable in the trees due to the large number of small triangles, but was also noticeable in the small detail of the front porch of the house. To solve this problem, we redefined the colors so that the variation within the colors was not as great and so that the very light hues were not used. This removed much of the flashing and resulted in a more realistic image.

The fall colors were made by redefining the tree colors from an assortment of greens to reds, yellows, and greens. The ground color was also changed from green to brown.

VIDEO AND FILM TECHNIQUES

A portion of this project involves the ability of the group to make images on film or tape. The actual production problems we encountered proved to be challenging in their own way.

The use of Polaroid SX-70 film and camera was necessary for documentation of our progress during the project and was helpful in describing successes or problems to the project director, Jack Booker. The Polaroid camera was used on a tripod and simply placed in front of a Ramtek RGB monitor (512x512). Although the photographs could not be claimed to be anything more than satisfactory, they were adequate for keeping a record of our progress during the different stages of the project.

For a more permanent record of the project a Pentax 35 mm camera was used with Ektachrome daylight film for slides. These color slides are a very accurate representation of the images that make up the terrain model itself and are very close in color and contrast to the final images in the animation. Once again, the camera was mounted on a tripod and placed in front of our Ramtek monitor in order to photograph the image. We feel the slides are nearly as important as the animation itself in revealing the complexity of the terrain model data base.

For completion of the animation portion of this project an Arrflex 16 mm camera was used with Ektachrome film. Since the animation frames were calculated and stored on disk or mag tape to be played back single frame, a routine was written which controls the camera so that as each

frame is displayed on the monitor, the camera shutter is activated and the entire sequence is filmed. The Arriflex was set with an F-stop of 5.6 and an exposure time of ½second. If a video tape is required it is possible to run the film through a film chain to render a finished product of video tape.

SPECIFIC PROBLEMS ENCOUNTERED

During the course of the project, certain problems arose; some expected, some of them totally unexpected. The following section is an account of the discovery of and solutions to these problems.

The initial endeavors of our group into the terrain model problem was somewhat tentative and there existed an inclination to tackle the problems therein with the tools already at our disposal. It soon became apparent that although we were experienced in the generation and display of 3-dimensional data, new methods would have to be developed in order to cope with the new assignment of terrain model generation. The first of these problems was dealing with trees as objects.

The sizing of trees was proving to be a road block. While a mountain or hill could be sized along the three axes the relative size of the trees was so enormous that the mountains were dwarfed in comparizon. A command is now incorporated into the animation script which enables the user to scale the trees before positioning them in three-space. The bills, mountains and buildings will all be sized before placement in the terrain. This assures accuracy for the user and avoids the problem of terrain features becoming distorted during an animation sequence.

The animators working on the project seemed to become entangled in the masses of data which made up the data base. A solution was reached through the use of a technique which was drawn directly from our animation story boarding technique (in which the animator roughly sketches out key frames of the animation onto a story board).

While constructing the data base for the terrain model a "map" was drawn on graph paper. The map is made up of a grid on which objects are positioned in relation to their actual positions in the data base. The coordinate system allows the user to refer to the map when plotting a path for the observer, thereby avoiding all the difficulties encountered when attempting to call up positions and sizes of objects in the terrain.

A unit/foot coordinate system has been worked out so that positioning as well as creating data will be greatly simplified. One unit equals six inches or two units equal one foot. The trees in the data base have been sized to correspond with the buildings in the scene.

The more complex the image the more calculation time we were faced with. This became a problem, especially from the standpoint of tree generation and display. The calculation time was reduced to some degree by the incorporation of a 300 megabyte disk into the system. More significantly, perhaps, was the decision to perform a transform on the individual trees, thereby storing and recalling them as single objects which, in turn, eliminates the need to generate each tree procedurally from frame to frame. The calculation time was roughly quartered for generating trees.

ALIASING

One can view a solution to the problem as considering each pixel an area sampler which performs a weighted averaging of all the different portions of objects which are partially visible to that pixel. Each pixel would then be considered a mini-screen onto which the objects in the three-dimensional space are projected and the visible surface algorithm is performed. The relative size of the pixel's area covered by the visible surfaces could then be used as a weight, to be used in averaging the colors of the visible surfaces to produce a single red-green-blue value for that pixel. In order for this process to be carried out, each pixel would have associated with it a list of surfaces visible to that pixel. Each surface in the list would have associated with it an area of the pixel that it covers. If the objects being processed by the visible surface algorithm are presorted, then any incoming object portions visible to the pixel can be laid over the areas already associated with the pixel. If, however, the objects are being processed in an unordered stream using on-the-fly z-depth calculations, tests for visibility must be made during this processing.

Some simplifications can be made at some sacrifice in accuracy. For example, the pixel, instead of being considered as a continuous area, can be considered to be made up of a number of discrete "sub-pixels". The area covered by incoming surfaces can then be discretized to consider only these locations. This is, in effect, using a higher-resolution frame buffer and then averaging down to the resolution of the display frame buffer. An alternative to this is to introduce some type of blurring effect by allowing some overlap in the sub-pixels of adjacent pixels. The advantage of this scheme is the simplification in the scanning code -- the processing is the same as the initial process with the introduction of some averaging code as a post-processing step. The disadvantage of this approach is the added scanning computation needed for even a simple scene, especially if that scene covers a large area of the screen.

Our present system was not designed to address the issue of anti-aliasing so that several difficulties arise when trying to fit such a scheme into the present processing. The current scanning algorithm creates an image by producing the pixels covered by an object with their

associated z-depth. If the depth is closer than the depth associated with the pixel in the frame buffer, then the color and depth is replaced in the frame buffer by that which was just generated. This complicates anti-aliasing because 1) visible portions of objects are not generated, just those pixels covered by the object, 2) stream processing is used, not sorted objects, 3) a palette-type frame buffer is the output device so that the average of two or more colors is not guaranteed to be in the palette. Restricted cases of anti-aliasing have been tried in which the average color (or one very close) is guaranteed to be in the palette have been tried (i.e., averaging between two values of the same hue and between a value of a hue and black) and the results have been promising.

CONCLUDING REMARKS

We feel that the implications for the kind of generation and display system are far reaching. One strength of the system is that it can be used for generating many different types of imagery. The terrain model problem offered a significant test of the system because of the various elements involved in the data base. A house and barn, for instance, required a much different approach to data generation than a stream or a road, and trees presented another problem entirely. The task was not limited to data generation, however. This data base of a terrain model (as mentioned in the intorduction) had to be animated to simulate a pilot's view from low flying aircraft and this provided yet more challenges to the CGRG as well as the animation system. The logical steps the group followed in order to meet the requirements of this project not only enabled us to complete the project, but also led us to new discoveries which may be applied to new projects in the future.

Procedure models for generating trees; B-spine interpolation for generating roads; grid routines for mountains, rivers and other terrain forms -- these all were part of the process of producing the final animation of the terrain model, but maybe more importantly, they give us the tools to try new approaches to problems of flight simulators and image enhancement.

Results of the research performed during the implementation of those techniques are going to be used by us in further investigations related to flight simulation. Efficient algorithms, for generating, displaying, and manipulating terrain data are essential for realistic imagery to be provided in real time. It might be that a marriage of techniques such as those described here and descriptive sources, like the Defence Mapping Agency data base, will provide some initial steps toward the realization of this efficiency.

SECTION II

TERRAIN MODEL PERFORMANCE TESTS ON Z-BUFFER ANTS SYSTEM

ANTS is a high performance, multi-user procedural based animation system developed at The Ohio State University by the Computer Graphics Research Group. It is used for producing highly detailed, color, 3-D animation that can be recorded on 16 mm film, 1" video tape, or stored for later recording. ANTS provides the facilities needed for producing a wide range of complex animation sequences. It has the ability to control a variety of data types such as points, lines and polygonal surfaces using a specially designed high level animation language. The language operates in a multi-level environment which facilitates the use of procedural modeling techniques by allowing multiple copies of the data to be created with a new set of spacial and display characteristics for each one. This is very useful for generating buildings or large terrain models.

The purpose of this evaluation is to measure three performance aspects of the ANTS animation system: execution speed, multi-environment, and virtual memory.

Test 1 shows how much time ANTS spends executing each of its individual tasks. Different versions of ANTS were made which selectively omitted several key program tasks from the normal flow of operation. By using each of these versions on the same data in the same environment, a breakdown of the individual task speeds can be reasonably deduced.

ANTS has three main sections to be timed: control, animation and display. Test 1 currently reflects only breakdowns of the control and display sections, because the animation section was not used to calculate the terrain scenes being tested. The control section is made up of parsing, object control and script control tasks. The control section generally takes up no more than 10% of the total time per frame, so its separate parts are treated as whole. The display section has two distinct parts: object to image space tasks, which consist of clipping, perspective, and color update. The object to image space tasks contain mostly mathematical execution routines and therefore tend to be CPU bound, while scanning routines involve mostly memory accesses to a disk based pixel buffer and therefore tend to be I/O bound.

Test 1 uses six versions of ANTS to determine the breakdowns of the the individual control and display tasks:

1) ANTS with all facilities to set the standard
2) ANTS without any display tasks to determine the control section speed
3) ANTS without the scanning routines to determine (with version #2), the speed for the total object to image space task.

4) ANTS without lightsource to get the combined time for the clipping and perspective time (using version #3). Note that clipping and perspective cannot be further broken down because they are both needed to properly calculate the terrain scenes.

5) ANTS without the z-compare per pixel and color update when pixel is found to be closest to observer. This version determines the speed of the tiler when used with version #3.

6) ANTS without the color update. This version only has one assembler instruction removed from the normal version #1. That instruction moves a 16 bit color word into appropriate pixel in a disk resident frame buffer.

   Since other tests conducted at CGRC have shown that one instruction more or less makes no difference to the overall speed of a large system such as ANTS, version #6 tests solely the amount of time spent executing the page faults incurred by accessing memory not currently found in physical memory.

Test 2 determines how the multi-user VMS VAX environment impacts the speed and efficiency of ANTS when running complex animation sequences. For this test, ANTS was run in several actual production environments. These were made up of users editing, assembling and linking programs, as well as running graphic programs, including other ANTS animation sequences and a CPU bound 'list priority scan-line' display algorithm. The 'list priority' algorithm is controlled by the ANTS control and animation sections, and is used for high quality image projects. Since ANTS spends a lot of time waiting for I/O, these tests indicate that two ANTS programs running together may be the best combination.

Test 3 shows how the VAX virtual memory system impacts performance. By changing working set sizes, the operating system varies the amount of physical memory any one program can have for paging in data from the disks. Since ANTS uses a z-buffer algorithm for displaying the terrain data, this number can be very important, because the color buffer and z-buffer are disk resident.

All tests have been run on a 32 bit VAX 11/780 using the virtual memory operating system, VMS 1.6. Special hardware includes a floating point accelerator, a 300 MB disk drive (used for storing animation frames), and a 512x512x10 bit (palette lookup) color frame buffer.

For the timing measurements, only the total CPU time and the total elapsed ('wall') time are taken. The 'page fault' measurements indicate how many times a page (½ kilobyte) had to be read from disk into physical memory.

The timings in these tests are averages of up to six samples, which vary about +/- 2% over both the CPU and elapsed times when there are no other users on the system. As the environment becomes more cluttered, the CPU time remains constant at about +/- 3%, but the elapsed (wall) time varies around a +/- 30% range.

VITAL VIEW STATISTICS

	VIEW 1	VIEW 2	VIEW 7	VIEW 11
FACES	27042	27042	27042	27042
EDGES	81126	81126	81126	81126
SCANNED FACES	21477	20156	9937	21090
SCANNED EDGES	64421	60468	29811	63270
Z-COMPARE	331150	348524	348281	295414
COLOR UPDATE	298164	309632	251512	281759

Using 'View 1' as a standard for comparison:

'View 2' scans 6% less faces than 'View 1' and does 5% more z-compares, with 4% more color updates.

'View 7' scans 54% less faces, but does 5% more z-compares, with 16% less color updates.

'View 11' scans 2% less faces, 12% less z-compares, and 6% less color updates than 'View 1'.

OVERALL 4 VIEW TIMINGS OF ANTS WITH ALL FACILITIES

	CPU TIME	ELAPSED TIME	PAGE FAULTS
VIEW 1			
AVER:	153 sec 2.6 min	239 sec 4 min	52341
VIEW 2			
AVER:	163 sec 2.7 min	278 sec 4.6 min	67054
VIEW 7			
AVER:	103 sec 1.7 min	141 sec 2.3 min	22618
VIEW 11			
AVER:	162 sec 2.7 min	285 sec 4.8 min	63818

Again using 'View 1' as a standard for comparison, and assuming a physical memory working set size of 256 pages, with no other users in the environment:

'View 2' takes 6% more CPU time, 14% more elapsed time, and 22% more page faults than 'View 1'. This is mainly due to an increase in z-compares and color updates.

'View 7' uses 33% less CPU time, 41% less elapsed time, and 56% less page faults. This is the result of a 54% decrease in actual scanned faces.

'View 11' uses 6% more CPU time, 16% more elapsed time, and 18% more page faults.

'View 11' scans less faces and performs less z-compares and color updates than 'View 1', but takes more elapsed time. This extra time seems to be the cause of 18% more page faults.

TEST 1 - 'ANTS' TASK TIMINGS AND PERCENTAGES

TABLE 1. VIEW 1 TIMINGS

ANTS: Physical Memory = 256 pages
Environment: No users

	CPU TIME	ELAPSED TIME	PAGE FAULTS
Tested ANTS : All facilities			
Aver:	153 sec 2.6 min	239 sec 2 min	52341
Tested ANTS: Without 'object to image' - clipping, perspective, light : Also without 'scanning' - tile, z-compare, color update			
Aver:	13 sec	20 sec	37 sec
Tested ANTS: Without 'scanning' - tile, z-compare and color update			
Aver:	93 sec 1.6 min	101 sec 1.7 min	43
Tested ANTS: Without 'z-compare and color update'			
Aver:	119 sec 2 min	126 sec 2.1 min	43
Tested ANTS: Without 'color update'			
Aver:	134 sec 2.2 min	163 sec 2.7 min	11478
Tested ANTS: Without 'lightsource'			
Aver:	140 sec 2.3 min	225 sec 3.8 min	51213

TABLE 2. VIEW 1 PERCENTAGES

	CPU TIME	ELAPSED TIME	PAGE FAULTS
All Facilities	100% (153 sec)	100% (239 sec)	100% (52341 pages)
ANTS Control	9% (14)	8% (19)	2% (1047)
All Scan	40% (61)	58% (139)	98% (51294)
Tile	17% (26)	11% (26)	0%
Z-Compare	10% (15)	15% (35)	22% (11515)
Color Update	13% (20)	32% (77)	76% (39779)
All Object to Image	51% (78)	34% (81)	0%
Lightsource	8% (12)	6% (14)	0%
Clip/Perspective	43% (66)	28% (67)	0%

'View 1' has many timing characteristics that are similar among the four views:

* The object to image space tasks and the tiler all have nearly the same CPU and elapsed time in seconds, indicating they are truly CPU bound.

* The z compare and color update tasks show a considerable difference between CPU and total elapsed time, in seconds, which shows their I/O dependency.

* The scanning tasks, z compare and color update have nearly 100% of the total system page faults due to their use of a disk based color and z pixel buffer.

* The relationship of the scanning routines to all other ANTS routines is about 40% of the total CPU time, but 60% the actual elapsed time.

Figure 1. View 1

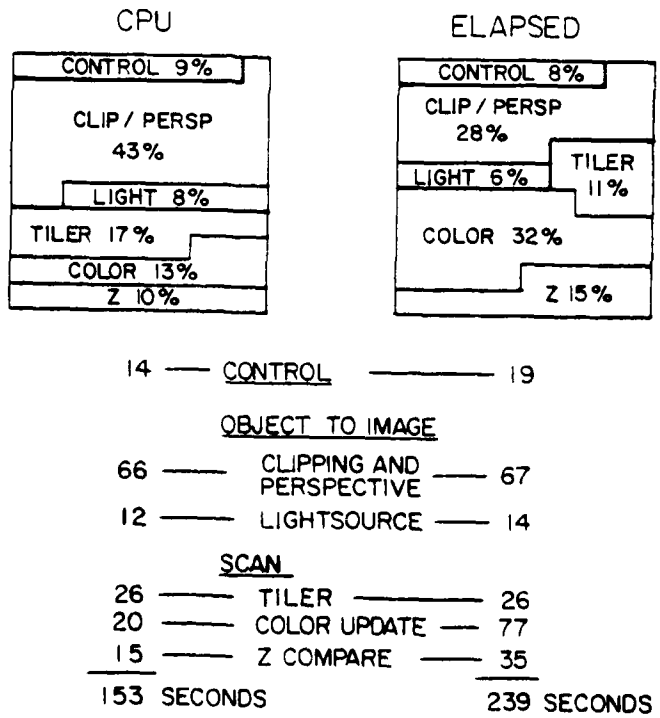


TABLE 3. VIEW 2 TIMINGS

ANTS: Physical Memory = 256 pages
Environment : No users

	CPU TIME	ELAPSED TIME	PAGE FAULTS
Tested ANTS: All facilities			
Aver:	163 sec 2.7 min	278 sec 4.6 min	67054
Tested ANTS: Without 'object to image' - clipping, perspective, light : Also without 'scanning' - tile, z-compare, color update			
Aver:	13 sec	20 sec	41
Tested ANTS: Without 'scanning' - tile, z-compare and color update			
Aver:	91 sec 1.5 min	99 sec 1.7 min	44
Tested ANTS: Without 'z-compare and color update'			
Aver:	119 sec 2 min	127 sec 2.1 min	43
Tested ANTS: Without 'color update'			
Aver:	136 sec 2.3 min	168 sec 2.8 min	19660
Tested ANTS: Without 'lightsource'			
Aver:	149 sec 2.5 min	266 sec 4.4 min	65468

TABLE 4. VIEW 2 PERCENTAGES

	CPU TIME	ELAPSED TIME	PAGE FAULTS
All Facilities	100% (163 sec)	100% (278 sec)	100% (67054 pages)
ANTS Control	8% (13)	7% (19)	2% (1341)
All Scan	44% (72)	64% (178)	98% (65713)
Tile	17% (28)	10% (28)	0%
Z-Compare	10% (16)	18% (50)	36% (24140)
Color Update	17% (28)	36% (100)	62% (41573)
All Object to Image	48% (78)	29% (81)	0%
Lightsource	9% (14)	5% (14)	0%
Clip/Perspective	39% (64)	24% (67)	0%

'View 2' is very similar to 'View 1' in that all faces of the terrain are being clipped and perspected, which accounts for the exact same elapsed times for those tasks. However, 'View 2' is from a much different angle so that paye faults are different between the two.

These figures indicate that the extra time used per frame over 'View 1' is spent in the z-compare and color update tasks.

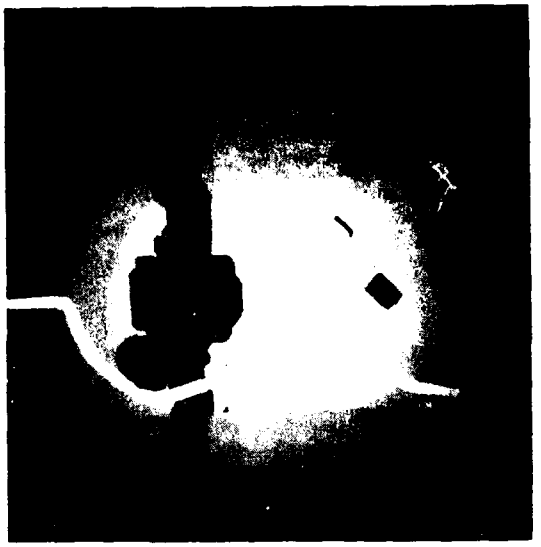
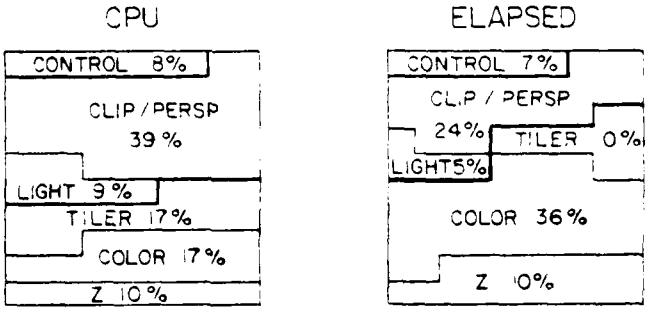


Figure 2. View 2



13	CONTROL	19
	OBJECT TO IMAGE	
64	CLIPPING AND PERSPECTIVE	67
14	LIGHTSOURCE	14
	SCAN	
28	TILER	28
28	COLOR UPDATE	100
16	Z COMPARE	50
163 SECONDS		278 SECONDS

TABLE 5. VIEW 7 TIMINGS

ANTS: physical Memory = 256 pages
Environment: No users

	CPU TIME	ELAPSED TIME	PAGE FAULTS
Tested ANTS: All facilities			
Aver:	103 sec 1.7 min	141 sec 2.3 min	22618
Tested ANTS: Without 'object to image' - clipping, perspective, light : Also without 'scanning' - tile, z-compare, color update			
Aver:	13 sec	20 sec	44
Tested ANTS: Without 'scanning' - tile, z-compare and color update			
Aver:	67 sec 1.1 min	75 sec 1.3 min	45
Tested ANTS: Without 'color update'			
Aver:	95 sec 1.6 min	119 sec 2 min	12467
Tested ANTS: Without 'lightsource'			
Aver:	97 sec 1.6 min	134 sec 2.2 min	22495

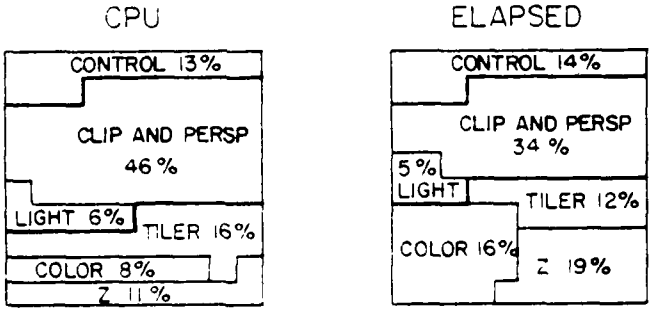
TABLE 6. VIEW 7 PERCENTAGES

	CPU TIME	ELAPSED TIME	PAGE FAULTS
All Facilities	100% (103 sec)	100% (141 sec)	100% (22618 pages)
ANTS Control	13% (13)	14% (20)	2% (452)
All Scan	35% (36)	47% (66)	98% (22165)
Tile	16% (17)	12% (17)	0%
Z-Compare	11% (11)	19% (27)	55% (12440)
Color Update	8% (8)	16% (23)	45% (10178)
All Object to Image	52% (54)	39% (55)	0%
Lightsource	6% (6)	5% (7)	0%
Clip/Perspective	46% (47)	34% (48)	0%

'View 7' is the fastest of the four views to calculate.

These figures show an increased control overhead percentage compared to 'View 1 and 2'. Also, the ratio of 'object to image space' tasks to the scanning routines is increased. This is because the decrease in page faults has allowed the CPU bound tasks to occupy a larger amount of the total elapsed time. It is also interested to note that the ratio of page faults between z-compare and color update has switched from the 25%-75% ratio in 'View 1' to a 55%-45% ratio in 'View 7'. This is because of an increase in the number of depths levels that the data has at this view angle.

Figure 3. View 7



13	CONTROL	20
	OBJECT TO IMAGE	
47	CLIPPING AND PERSPECTIVE	48
6	LIGHTSOURCE	7
	SCAN	
17	TILER	17
8	COLOR UPDATE	23
11	Z COMPARE	27
103 SECONDS		141 SECONDS

TABLE 7. VIEW 11 TIMINGS

	CPU TIME	ELAPSED TIME	PAGE FAULTS
Tested ANTS: All facilities			
Aver:	162 sec 2.7 min	285 sec 4.8 min	63818
Tested ANTS: Without 'object to image' - clipping. perspective, light ; Also without 'scanning' - tile, z-compare, color update			
Aver:	13 sec	20 sec	41
Tested ANTS: Without 'scanning' - tile, z-compare and color update			
Aver:	94 sec 1.6 min	102 sec 1.7 min	42
Tested ANTS: Without 'z-compare and color update'			
Aver:	121 sec 2 min	129 sec 2.2 min	45
Tested ANTS: Without 'color update'			
Aver:	133 sec 2.2 min	162 sec 2.7 min	15905
Tested ANTS: Without 'lightsource'			
Aver:	147 sec 2.5 min	274 sec 4.6 min	62635

TABLE 8. VIEW 11 PERCENTAGES

	CPU TIME	ELAPSED TIME	PAGE FAULTS
All Facilities	100% (161 sec)	100% (285 sec)	100% (63818 pages)
ANTS Control	8% (13)	7% (20)	2% (1276)
All Scan	42% (68)	64% (182)	98% (62542)
Tile	17% (28)	9% (26)	0%
Z-Compare	7% (11)	12% (34)	25% (15955)
Color Update	18% (29)	43% (123)	75% (47863)
All Object to Image	50% (81)	29% (83)	0%
Lightsource	9% (14)	4% (12)	0%
Clip/Perspective	41% (67)	25% (71)	0%

'View 11' is the longest of all the views to calculate.

These figures indicate that whatever is taking longer seems to be caused by the color update section. It appears that even though 'View 11' has less actual scanned faces and less z compares and color updates, it still can take longer to calculate if the view is oriented in such a way that accessing the pixel buffer caused extra page faults, which this view does.

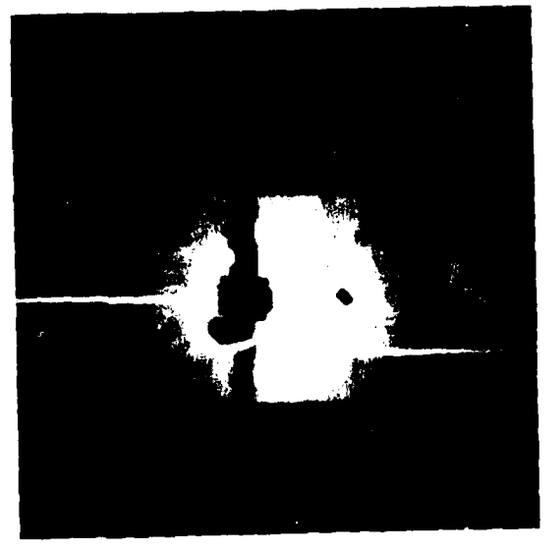
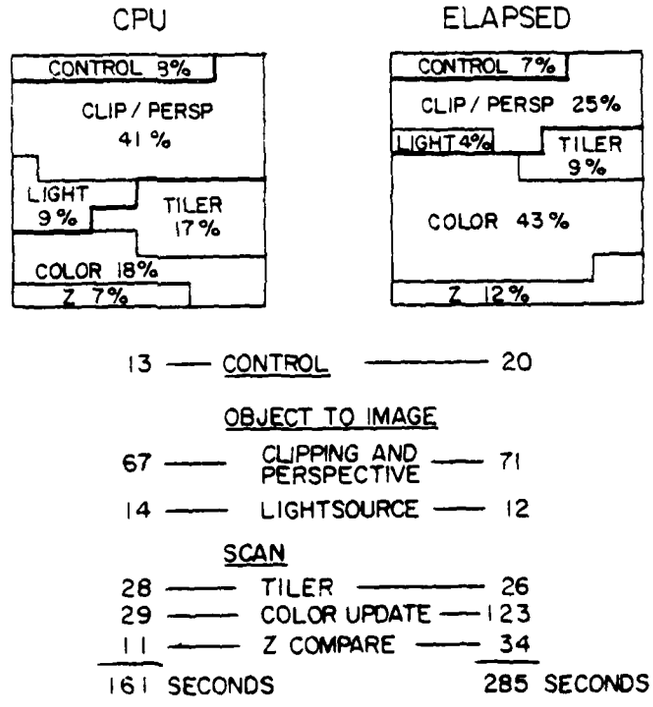


Figure 4. View 11



CONCLUSION FOR TEST 1

These timing tests of four different views of the same terrain data are not a complete breakdown of all the separate tasks of ANTS, but they do show some important relationships concerning the most time critical (and time consuming) tasks within the display section. The control section timings are very consistent throughout all four views, because basically the same code is executed regardless of the view being calculated. These timings are also only a small percent of the overall elapsed time of any view, about 20 seconds for handling 27000 faces, which is good compared with 3-5 minutes of elapsed time per frame. The display section, however, is very image dependent, and varies considerably over the four views, usually accounting for about 90% of the total frame time. This section has two distinct sub-tasks: an object to image space task, and a face scan to frame buffer task. Generally, about 40% of the total CPU time is spent scanning, while the object to image tasks use about 50%. This relationship flips, however, when the total elapsed time is considered. Now the scanning tasks occupy around 60% of the time while the object to image tasks only take up about 35%. This is because the object to image space tasks are almost totally compute bound, and slow down very little when comparing CPU to elapsed time. The scanning routines are very I/O dependent, depending on disk resident pages to be read into physical memory to cover the areas of the frame buffer being accessed. The elapsed time is 45% more than CPU time within these 4 views alone.

Clearly, if some method were available for eliminating the page faulting, ANTS would run more efficiently and be much more predictable. Currently the scanning routines account for 98% of all page faults. These are divided between doing z-compares and doing color updates, with the lion's share seemingly going to the color update task. In fact, this measurement is misleading. It indicates that the VAX VMS paging system is reasonably good if given an ideal situation. The problem is that we currently store our pixel buffer as two distinct data areas: a 512x512x16 bit color buffer, and a 512x512x32 bit z (depth) buffer. From the timings in Test 1, it appears that if one of these two disk resident buffers are eliminated, then the VMS memory paging system seems to work very well, producing, for example, a 50% drop in total elapsed time in 'View 1'. It is important to note that eliminating the color update which caused this decrease consisted only of removing one assembler command, a 'move' of a color word into the appropriate pixel in the color buffer. By not doing this, the color buffer never has to be paged into memory, leaving the z-buffer all to itself in the ANTS working set. Completely eliminating all frame buffer accesses only decreases the elapsed time by another 20%, so it seems to follow that VMS can handle one very large virtual data area fairly well, but not two.

The solution to speeding up the display section is twofold. First, the entire pixel buffer, both color and depth, must be put into fast semiconductor memory and be made directly addressable to the 'ANTS' tasks. The addressability is important, for without it a lot of time is lost communicating between the VAX and the frame memory store. The second enhancement would be to do all the calculations for the object to image space tasks in a fast bit sliced microcomputer, taking the burden off the comparatively slower general purpose VAX. Ideally the output of the micro would be pipelined to another bit sliced microprocessor which would do the scanning task and directly update the hardware pixel buffer. Such a solution could be expected to reduce the total elapsed time by 80-90%.

The most important timings not yet availabie are those concerning the ANTS animation section. These include a parsing and control task, and an animation transform task for performing rotation, position, and size changes. While control tasks typically have a very low impact on the total elapsed time of a frame, animation requiring complex motions of many separate objects can cause the transform task to chew up a significant amount of the elapsed time per frame. Also unavailable from Test 1 are breakdown timings of ANTS when run with other users in the environment running other various tasks. This would show how the individual tasks of ANTS are affected, as shown in Test 2, which measures ANTS in different environments.

NAVTRAEQUIPCEN 80-C-0008-1

TEST 2

TESTS ANTS IN A MULTI-USER ENVIRONMENT

VIEW 1 TIMINGS

Tested ANTS : All facilities run at priority 4
: Physical memory = 256 pages

Environment : Physical memory = 256 pages

	CPU TIME	ELAPSED TIME	PAGE FAULTS
VIEW 1			
ENVIRONMENT: No users			
AVER:	153 sec	239 sec	52341
	2.6 min	4 min	

COMMENT: This environment is used as the standard for comparison.
ANTS CPU time accounts for 64% of total time.

VIEW 1			
ENVIRONMENT: 1 User - reading mag tape and recording on 1 inch video			
AVER:	157 sec	250 sec	52336 pages
	2.6 min	4.2 min	

COMMENT: This environment causes ANTS to run 5% slower.
CPU time accounts for 63% of total.

VIEW 1			
ENVIRONMENT: 1 Animation 'list' at priority 4			
AVER:	154 sec	346 sec	53098 pages
	2.6 min	5.8 min	

COMMENT: This environment causes ANTS to run 45% slower.
CPU time accounts for 45% of total.

```
VIEW 1
ENVIRONMENT:  1 Animation 'list' at priority 4
ENVIRONMENT:  1 Animation 'z-compare' at priority 3
AVER:                  155 sec          369 sec          52041 pages
                       2.6 min          6.2 min

COMMENT:  This environment causes ANTS to run 54% slower.
          CPU time accounts for 42% of total.


VIEW 1
ENVIRONMENT:  1 Animation 'list' at priority 4
ENVIRONMENT:  1 Animation 'z-compare' at priority 3
ENVIRONMENT:  2 Users - general editing, file transfer, light ASM/BLD
AVER:                  153 sec          387 sec          52159 pages
                       2.6 min          6.5 min

COMMENT:  This environment causes ANTS to run 62% slower.
          CPU time accounts for 40% of total.


VIEW 1
ENVIRONMENT:  1 Animation 'list' at priority 4
ENVIRONMENT:  1 Animation 'z-compare' at priority 3
ENVIRONMENT:  1 User - running priority 9 program with heavy I/O
AVER:                  159 sec          422 sec          52059 pages
                       2.7 min          7.0 min

COMMENT:  This environment causes ANTS to run 77% slower.
          CPU time accounts for 38% of total.


VIEW 1
ENVIRONMENT:  1 Animation 'list' at priority 4
ENVIRONMENT:  1 Animation 'z-compare' at priority 3
ENVIRONMENT:  1 User - doing long assembly/task-build at PRI 7/8/9
AVER:                  170 sec          545 sec          50903 pages
                       2.8 min          9.1 min

COMMENT:  This environment causes ANTS to run 128% slower.
          CPU time accounts for 31% of total time.
```

CONCLUSION FOR TEST 2

Test 2 shows timings of an identical version of ANTS run under a variety of multi-user environments. As might be expected, these tests show that as the environment becomes more cluttered with other users, the performance of the tested ANTS decreases. Note that the CPU time remains very consistent over each of the tests, but the total elapsed time increases relative to the complexity of the tasks co-sharing the CPU environment. This is because the extra time it takes to run ANTS under these different environments is coming from the I/O bound routines and note the CPU bound routines. In other words, each time ANTS comes to an I/O wait, its execution will not be resumed until one of the other tasks also being executed comes to an I/O wait. Therefore each I/O request from ANTS can take considerably longer to perform in a multi-user environment, than when ANTS is running by itself.

An interesting finding in this test is the increase in elapsed time in a multi-user environment is not simply a multiple of the time it takes running alone. For example, if ANTS takes four minutes to run by itself, then one might initially assume that two copies of ANTS running together would take eight minutes, but in fact they seem to take only six or seven minutes together. This is largely because when one ANTS program is waiting for I/O, it does not need any CPU resource, so another program can be calculating something while it waits. Therefore even though the total elapsed time might be more, the total time for two ANTS programs will be smaller than if run separately, because the system resources are being more efficiently used. This condition can be even more useful and efficient if one of the ANTS programs was more CPU bound than the other, and run at a lower priority. This would help prevent repetitious task switching back and forth between the two programs, causing valuable CPU time to be wasted.

TEST 3

VIRTUAL MEMORY TESTS

Tested ANTS ; All facilities run a priority 4

	CPU TIME	ELAPSED TIME	PAGE FAULTS
VIEW 1			
TESTED ANTS:	Physical memory = 256 pages		
ENVIRONMENT:	No users		
AVER:	153 sec	239 sec	52341
	2.6 min	4 min	

COMMENT: This environment is used as the standard for comparison. ANTS CPU time accounts for 64% of total time.

VIEW 1			
TESTED ANTS:	Physical memory = 512 pages		
ENVIRONMENT:	No users		
	133.2 sec	216.6 sec	10437 pages
	135.8 sec	218.9 sec	10451 pages
	269	435.5	20888
AVER:	135 sec	218 sec	10444 pages
	2.2 min	3.6 min	

COMMENT: Increasing the working memory set to 512 pages causes ANTS to run 9% faster than standard total time. It also causes an 80% decrease in the number of page faults.
CPU time accounts for 62% of total time.

VIEW 1			
TESTED ANTS:	Physical memory = 768 pages		
ENVIRONMENT:	No users		
	129.5 sec	208.1 sec	5882 pages
	132.9 sec	212.2 sec	5879 pages
	262.4	420.3	11761
	131 sec	410 sec	5881 pages
	2.2 min	3.5 min	

COMMENT: Increasing the working memory set to 768 pages causes ANTS to run 12% faster than standard total time. It also causes an 89% decrease in the number of page faults.
CPU time accounts for 62% of total time.

VIEW 1
TESTED ANTS: Physical memory = 1024 pages
ENVIRONMENT: No users

128.8 sec	204.1 sec	4361
2.1 min	3.4 min	

COMMENT: Increasing the working memory set to 1024 pages causes ANTS to run 15% faster than standard total time. It also causes a 92% decrease in the number of page faults.
CPU time accounts for 63% of total time.

CONCLUSION FOR TEST 3

Test 3 shows the effects of changing the working set size in the ANTS programs. The size parameter determines how much physical memory the system can use to page data in from the disk. The more physical memory employed, the less the system has to write over old page accesses, which means there is a greater chance of finding something wanted from the disk already read into main memory.

This test is still a bit of a puzzle. One would expect an increase in the working set to cause a decrease in the number of page faults required to calculate a given scene. This is in fact exactly what happens. A simple doubling of the working set size from 256 pages to 512 pages for example, causes an 80% decrease in page faults. What is puzzling is that one would further expect a considerable decrease in the amount of elapsed time, which would correspond the drop in page faults, but this doesn't happen. Instead of a 30 to 40% decrease in time, based on the findings in Test 1, we get only a 9 to 15% drop. While this is a fairly good improvement, we are still performing tests to determine the source of this displexity.

One initial clue seems to be that two ANTS programs running together can be made more efficient by using different working set sizes. This indicates that even with a high size parameter, time is still being wasted somewhere which can be better used by another task. It may be that although there are fewer pages faults, they might be taking longer to do individually, which would minimize the gains resulting from the memory size increase.

CONCLUSION FOR ALL THREE TESTS

These tests have shown various performance measurements of the ANTS animation system under the operating system VAX/VMS. Test 1 showed a breakdown of the execution speeds of the different display tasks while they calculated four different views of the same terrain data. Test 2 showed how the same version of ANTS behaved when placed in different user environments, and Test 3 showed how it performed with different virtual memory parameters.

Test 1 mainly illustrated the distinction between the CPU bound tasks of tiling, clipping, perspective and lightsource, and the I/O bound scanning tasks. The CPU bound tasks typically showed an increase of less than 5% from the CPU time to the total elapsed time, while the scanning routines usually increased in time by at least 100%.

It was this increased time of the scanning routine which was the more affected by the system and environment changes of Tests 2 and 3.

Here we saw an increase in the scanning tasks as more users were added to the environment, and a decrease in time when ANTS's physical memory size was increased.

While most of the relationships of the tests shown were anticipated before this exercise began, there were several results that are still unexplained. These uncertainties largely revolve around our unfamiliarity with the precise manner in which the VAX/VMS paging system works. For example, we do not know why at a 92% decrease in page faults only resulted in a 15% decrease of the total elapsed time could be seen when the page faults were lowered by 98% by simply eliminating all disk-based frame buffer accesses in Test 1. Also, it is unclear why in recent tests not covered in this report, that we can run 2 ANTS programs together and they indicate that they have taken 2 minutes each of elapsed time, but a stop watch shows they have only taken 3 minutes or less total.

Understanding these puzzles better will help us to fine tune the VAX system environment to squeeze more performance out of ANTS without resorting to any drastic measures, such as rewriting software. The most immediate benefit resulting from these and other tests is that it seems that with some small adjustments to the working set size and the paging system, we should be able to run two ANTS programs for the time price of one. This is an extremely attractive option, because it means that we can speed up ANTS by 100% without making any software changes either to ANTS, or to the operating system.

The tests also show that by making some minor redesigns within the ANTS scanning tasks, we can get even better performance out of ANTS beyond just running two programs at once. These changes include:

* Using only one data structure to define the pixel buffer rather than the two used now. Currently we keep a 512x512x16 bit color buffer followed by a 512x512x32 bit depth buffer on the disk. Test 1 seemed to show that VAX/VMS is very efficient at paging in one large data area into physical memory, but not two. They seem to work against one another by overwriting the other's pages already read in.

* Not using a disk-based color buffer at all. This could be done by writing out to the hardware color frame buffer currently used for display. It is not certain, however, whether the 'QUOs' involved in this access will be significantly faster than just paging in disk memory.

* Locking both the color buffer and the depth buffer into physical memory. This would mean that each ANTS program would consume a very large amount of VAX memory resources, so only a few users could be on the system at once, but the increase in speed could

justify that problem. This however, might not be much of a performance increase from simply increasing the working set size to the largest value VMS can hold and letting the VMS paging system continue from there.

* Using smaller pixel buffers rather than the standard full screen 512x512 one which we use now. By dynamically allocating a frame buffer just large enough to scan the object being processed, it would be possible to create a data structure small enough to be locked completely into main memory while the scanning took place. The mini-buffer could then be merged into a full screen pixel buffer for the final image. This process could be particularly beneficial for scanning trees, which are typically small but are densely populated with small triangles.

These performance modifications and a new, more complete set of timings will be presented more fully in a later report.

APPENDIX A

TIMINGS FOR ANTS IN TEST 1

TEST 1

Tested ANTS: All Facilities
: Physical Memory = 256 pages
Environment: No Users

	CPU TIME SECONDS	ELAPSED TIME SECONDS	PAGE FAULTS PAGES
VIEW 1:	153.5	235.3	52086
	153.4	235.9	52154
	155.6	238.8	52081
	153.4	237.7	52164
	149.8	233.4	52280
	152.5	243.0	52857
	154.7	245.5	52766
	1072.9	1669.6	366388
AVER:	153	239	52341
VIEW 2:	165.0	279.5	66829
	163.6	275.4	67120
	161.3	278.8	67213
	489.9	833.7	201162
AVER:	163	278	67054
VIEW 7:	104.5	142.1	22619
	102.6	140.2	22618
	102.8	140.8	22620
	102.0	140.6	22620
	411.9	563.6	90475
AVER:	103	141	22618
VIEW 11:	163.1	285.7	63832
	160.6	283.5	63769
	161.1	284.9	63853
	484.8	854.1	191454
AVER:	161.6	284.7	63818

Tested ANTS: Without 'Lightsource'
: Physical Memory = 256 pages
Environment: No Users

	SECONDS	SECONDS	PAGES
VIEW 1:	140.1	225.4	51228
	140.8	224.5	51199
	280.9	449.9	102427
AVER:	140	225	51213
VIEW 2:	149.4	267.2	65473
	149.7	265.5	65463
	299.1	532.7	130936
AVER:	149	266	65468
VIEW 7:	97.3	134.2	22797
	96.6	132.9	22494
	193.9	267.1	44991
AVER:	97	134	22495
VIEW 11:	145.7	272.2	62641
	148.2	275.3	62629
	293.9	547.5	125270
AVER:	147	274	62635

Tested ANTS: Without 'Scan, Z-compare and Color Update'
: Physical Memory = 256 pages
Environment: No Users

VIEW 1:	92.4	99.8	41
	93.3	101.3	45
	185.7	201.1	86
AVER:	93	101	43
VIEW 2:	91.1	98.3	43
	91.5	99.7	46
	182.6	198.0	89
AVER:	91	99	44
VIEW 7:	66.9	74.7	45
	67.3	75.2	45
AVER:	67	75	45

	SECONDS	SECONDS	PAGES
VIEW 11:	93.3	101.8	42
	93.9	102.0	43
	187.2	203.8	85
AVER:	94	102	42

Tested ANTS: Without 'Clip, Perspective, Light, Scan, Z-compare, Color Update'
: Physical Memory = 256 pages
Environment: No Users

VIEW 1:	12.6	19.8	35
	12.6	20.3	40
	25.2	40.1	75
AVER:	13	20	37
VIEW 2:	12.5	20.3	42
	12.7	20.3	40
AVER:	13	20	41
VIEW 7:	12.5	20.4	40
	12.6	20.4	45
AVER:	13	20	44
VIEW 11:	12.4	19.6	40
	12.6	20.4	41
	25.0	40.0	81
AVER:	13	20	41

Tested ANTS: Without 'Z-compare and Color Update
: Physical Memory = 256 pages
Environment: No Users

VIEW 1:	118.4	125.8	42
	118.7	126.8	45
	237.4	252.6	87
AVER:	119	126	43
VIEW 2:	119.6	127.6	41
	118.8	127.3	45
	238.4	254.9	86
AVER:	119	127	43

	SECONDS	SECONDS	PAGES
VIEW 7:	82.6	89.9	55
	83.2	91.1	45
	165.8	181.0	100
AVER:	82.9	90.5	50
VIEW 11:	120.6	128.6	46
	122.3	130.2	45
	242.9	258.8	91
AVER:	121	129	45

Tested ANTS: Without "Color Pixel Buffer Update'
: Physical Memory = 256 pages
Environment: No Users

VIEW 1:	132.9	162.2	11483
	135.0	164.7	11473
	267.9	326.9	22956
AVER:	134	163	11478
VIEW 2:	136.9	168.8	19639
	136.3	167.7	19682
	273.2	336.5	39321
AVER:	136	168	19660
VIEW 7:	94.9	118.7	12467
	95.5	119.8	12467
	190.4	238.5	24934
AVER:	95.2	119.2	12467
VIEW 11:	133.7	161.5	15904
	132.9	161.5	15906
	266.6	323.0	31810
AVER:	133	162	15905

APPENDIX B

TIMINGS FOR ANTS IN TEST 2

VIEW 1 TIMINGS

Tested ANTS: All Facilities
: Physical Memory = 256 pages

	CPU TIME (SECONDS)	ELAPSED TIME (SECONDS)	PAGE FAULTS (PAGES)
ENVIRONMENT: 1 Animation 'List at Priority 4			
VIEW 1:	152.1	339.3	53187
	153.6	350.7	52886
	154.7	349.9	53126
	155.4	344.8	53191
	615.8	1384.7	212390
AVER:	154	346	53098
ENVIRONMENT: 1 Animation 'List' at Priority 4			
ENVIRONMENT: 1 Animation 'Z-compare' at Priority 3			
VIEW 1:	153.4	365.5	51922
	154.5	[illegible].2	52143
	155.1	3[illegible]	52126
	155.1	371.2	51971
	618.1	1474.5	208162
AVER	155	369	52041
ENVIRONMENT: 1 User - Reading Mag Tape and Recording on 1 Inch Video			
VIEW 1:	157.2	252.3	52070
	152.9	242.3	52058
	156.5	256.9	52304
	156.6	253.2	52862
	159.4	258.3	52584
	159.8	238.7	52135
	156.9	249.4	52340
	1099.3	1751.1	366353
AVER:	157	250	52336

	SECONDS	SECONDS	PAGES
ENVIRONMENT:	1 Animation 'List' at Priority 4		
ENVIRONMENT:	1 Animation 'Z-compare' at Priority 3		
ENVIRONMENT:	2 Users - General Editing, File Transfer, Light ASM/BLD		
VIEW 1:	152.8	404.9	52183
	153.6	384.1	52181
	153.4	371.0	52113
	459.8	1160	156477
AVER:	153	387	52159
ENVIRONMENT:	1 Animation 'List' at Priority 4		
ENVIRONMENT:	1 Animation 'Z-compare' at Priority 3		
ENVIRONMENT:	1 User - Running Priority 9 Program with Heavy I/O		
VIEW 1:	159.1	420.9	51968
	158.7	422.8	52149
	317.8	843.7	104117
AVER:	159	422	52050
ENVIRONMENT:	1 Animation 'List' at Priority 4		
ENVIRONMENT:	1 Animation 'Z-compare' at Priority 3		
ENVIRONMENT:	1 User - Doing Long Assembly/Task-Build at Priority 7/8/9		
VIEW 1:	169.7	548.9	50065
	171.0	540.5	51740
	340.7	1089.4	101805
AVER:	170	545	50903

NAVTRAEQUIPCEN 80-C-0008-1

DISTRIBUTION LIST

Naval Training Equipment Center
Orlando, FL 32813

Defense Technical Information Ctr (12)
Cameron Station
Alexandria, VA 22314

Ohio State University
Computer Graphics Research Group
1314 Kinnear Road
Columbus, Ohio 43212

PM TRADE
Attn: RE
Orlando, FL 32813

ALL OTHERS RECEIVE ONE COPY

Chief of Naval Educ & Training
Code N331
Pensacola, FL 32508

Naval Air Systems Command
AIR 340D
Washington, DC 20350

Naval Air Systems Command
AIR 340F
Washington, DC 20350

Naval Air Systems Command
AIR 360
Washington, DC 20350

Naval Air Systems Command
Library AIR 50174
Washington, DC 20350

Chief of Naval Material, Navy Dept
Code MAT 08T
Washington, DC 20360

Chief of Naval Material, Navy Dept
Code MAT 08Y
Washington, DC 20360

Chief of Naval Material, Navy Dept
Code 09Y
Washington, DC 20360

Research Triangle Institute
PO Box 12194 (Attn: Jorge Montoya)
Research Triangle Park, NC 27709

(25) Chief of Naval Research
ONR 461
800 N. Quincy St.
Arlington, VA 22217

Naval Air Systems Command
AIR 413
(10) Washington, DC 20350

AF ASD/ENE
Attn: Mr. A. Doty
Wright-Patterson AFB, OH 45433
(2)

AF ASD/ENE
Attn: Mr Mike Nicol
Wright-Patterson AFB, OH 45433

Chief of Naval Operations
Attn: OP 151
Washington, DC 20350

AFHRL
Attn: Warren Richeson
Williams AFB, AZ 85224

Scientific Tech Info Office
NASA
Washington, DC 20546

AF ASD/ENETV
Mr James D. Basinger
Wright-Patterson AFB, OH 45433

Dean H. Owen
Aviation Psychology Laboratory
Ohio State University
464C West 17th Ave
Columbus, OH 43210

DARPA/SSD
MAJ Jack A. Thorpe
1400 Wilson Blvd
Arlington, VA 22209

DMAAC
Attn: PRRN (R. Pierce)
St Louis AFS, MO 63118

HQ DMA
Attn: Howard Carr
Mass Ave at 34th St NW (Naval Obs)
Washington, DC 20390